Global Reduction in Wormhole k-ary n-cube Networks with Multidestination Exchange Worms[1]

Dhabaleswar K. Panda

Dept. of Computer and Information Science
The Ohio State University, Columbus, OH 43210-1277
Tel: (614)-292-5199, Fax: (614)-292-2911
E-mail: panda@cis.ohio-state.edu

Abstract: This paper presents a new approach to implement global reduction operations (including barrier synchronization) in wormhole k-ary n-cubes. The novelty lies in using *multi-destination* message passing mechanism instead of single destination (unicast) messages. Using pairwise *exchange* worms along each dimension, it is shown that global reduction and barrier synchronization operations, as defined by the Message Passing Interface (MPI) standard, can be implemented with n communication start-ups as compared to $2n\lceil\log_2 k\rceil$ start-ups required with unicast-based message passing. This leads to an asymptotic improvement by a factor of $\lceil 2log_2k\rceil$. For different values of communication start-up time and system size, the analysis indicates that the new framework can implement fast global reduction compared to the unicast-based scheme. Issues related to the new approach are studied and the required architectural modifications to the router interface are presented. The analysis indicates that as the communication start-up time continues to decrease in new generation systems, the proposed framework can be used for implementing fast global reduction and barrier synchronization in large wormhole-routed systems without a separate control network.

Keywords: Global Reduction, Collective Communication, Wormhole Routing, Virtual Cut-through, Broadcasting, Meshes, k-ary n-cubes, and Path-based Routing.

[1]This research is supported in part by the National Science Foundation Grant # MIP-9309627.

Contents

1 Introduction

The wormhole-routing switching technique is becoming the trend in building future parallel systems due to its inherent advantages like low-latency communication and reduced communication hardware overhead [8, 19]. Intel Paragon, Cray T3D, Ncube, J-Machine, and Stanford DASH are representative systems falling into this category. Such systems with direct interconnections are being used for supporting either distributed-memory or distributed-shared memory programming paradigms. In order to support these paradigms, the systems need fast communication and synchronization support from the underlying network [16]. The Message Passing Standard [18] has recently defined the importance of *collective communication* [17]. One important category in this class is *global reduction* (sum, max, min, or user-defined functions) where there is involvement from all processes of an user-defined group. As defined by the standard, the results may be available to only one member of the group or all the members. The operations can be carried on either scalar or vector data. Barrier synchronization [11, 12, 20, 24] is a special case of this class of operation where there is no data (just an event) and the result is available to all members of the group. Hence, in this paper, without loss of generality, we identify both reduction and barrier synchronization as a single class of reduction operation.

Many software schemes have been recently proposed in the literature to efficiently implement reduction[2, 9] and barrier synchronization[26] operations on wormhole-routed systems. All these schemes use multiple phases of point-to-point unicast message passing and encounter long latency due to multiple communication start-ups. Systems like Cray T3D and CM-5 use dedicated tree-based networks to provide fast global reduction and barrier synchronization. However, these schemes are not physically scalable. As the system size increases, more number of registers, address/data bits, tree-levels, and physical wires are required to implement these schemes. Hence, these control networks need to be redesigned with additional circuitry as the system scales, unless special care has been taken during the design phase of the system. This raises a question whether fast reduction and barrier synchronization can be implemented on direct networks using software message passing with architectural supports associated to each router. This will alleviate the need for a separate control network and can provide easy scalability as the system size grows.

Traditionally, the wormhole-routed systems have supported only *point-to-point* (unicast) message passing mechanism [19]. This mechanism allows a message to have only a single destination. Using unicast-based *send* and *receive* message passing primitives, reduction and barrier synchronization operations can easily be achieved using a two-step procedure: *gather* and *broadcast*. During the *gather* step, data/information is gathered through phases of upward tree communication. At the end of this step, reduced data/information is available with a single (root) processor. The second step involves broadcasting data to other processors in multiple phases using a downward tree communication.

For a k-ary n-cube system with k^n processors, this scheme requires $2n \log_2 k$ communication phases. The cost for each of the above communication phases on wormhole-routed systems is $(t_s + (d+l-1) \times t_p)$. The component t_s is the communication startup time, t_p is the propagation time per hop, and d is the distance between the source and destination processors in the communication phase, and l is the data size. Since $t_s \gg t_p$ under the current technology [6, 15, 25], the cost of each phase is dominated by t_s and the overall cost is $\propto$ to the number of communication phases. This makes the global reduction cost quite high. For example, with 5.0

microseconds communication start-up time, it will take at least 100.0 microseconds (ignoring data propagation and computation time) to perform reduction operation on 1K processors in a 32×32 system. Recently it has been reported by Barnett [1] that reduction operation on 8 bytes of data on 16x32 Intel Paragon takes around 7600 microseconds using their new InterCom library. This raises question whether efficient mechanisms are possible to implement fast reduction operation.

With a closer look, it can be seen that the above two-step procedure has remain unchanged since the first generation of multicomputers were developed using packet-switched interconnection. As the switching technology has advanced towards the third generation wormhole-routed switching with virtual channels and virtual cut-through flow control [7], the challenge leads to whether any intelligent mechanism blending well with the switching technique can be devised to provide fast global reduction through message-passing. The communication start-up time in current generation systems is also gradually reducing. Hence, if a mechanism can be developed to achieve reduction with less number of communication start-ups then the overall cost of reduction operations will be low and future systems need not support a separate control network.

In this paper, we take such a challenge in proposing a new approach to implement global reduction in k-ary n-cube wormhole networks. Instead of using single-destination unicast worm, we propose using a new type of multidestination *exchange* worms. The *multidestination* wormhole mechanism, introduced by us recently in [23] for efficient broadcasting and multicasting support, allows a message to propagate through paths conforming to the base-routing of a system. In this paper we use only single-dimensional paths, which always conform to e-cube, planar-adaptive [5], or fully-adaptive [10] schemes. These worms use a special bit-string encoding of multidestination addresses as proposed by Chiang and Ni in [4] to reduce the header length. During a reduction operation, since an incoming message from a neighbor processor may have to wait at a router interface for its associated processor to be ready with its data, we suggest a virtual cut-through technique together with a number of fixed size buffers at the router interface to ensure deadlock-freedom.

Based on the required architectural modifications and supports, we present algorithms for complete (all processors participating) global reduction. Reduction with arbitrary number of participating processors can be done with multidestination *gather* and *broadcast* worms, similar to the schemes outlined in [22] for barrier synchronization. Hence we only emphasize on complete global reduction in this paper. Using *exchange* worms, it is demonstrated that the operations can be implemented with n communication start-up for k-ary n-cube systems when the data size is smaller than the individual buffer size available at a router. For data size larger than the buffer size, we present a *pipelined* algorithm. Analytical results for cost complexity are developed and compared with the unicast-based scheme by taking into account link propagation delay, node delay, data size, and computation time needed at intermediate routers. The effect of buffer size on the performance of exchange worm-based algorithms is studied for 2D and 3D systems. The new scheme is also shown to be scalable for increasing system and data size.

The paper is organized as follows. An overview of the multidestination mechanism together with its path-based propagation model is presented in section 2. The exchange worm is introduced and used in section 3 to illustrate global reduction on a linear array. In section 4, we present algorithms for complete global reduction on k-ary n-cube systems. Analytical modeling of cost complexity are developed and evaluated in section 5. Finally, we conclude the paper with future research directions.

2 Multidestination Wormhole Routing

In this section, first we summarize wormhole message passing mechanism with multiple destinations [23]. Then we show how multidestination messages can be effectively routed in a wormhole network using available paths conforming to the base routing. Intrinsic benefits of multidestination mechanism over unicast message passing are demonstrated. Finally, the bit-destination encoding scheme to reduce the header size of a multidestination worm is discussed.

2.1 The Mechanism

In single destination wormhole message-passing, every message consists of a body and a header with the destination number. For a multidestination message, the header consists of multiple destinations and can span multiple flits. The *sender* node creates the list of destinations as an ordered list, depending on their intended order of traversal, and incorporate it into the header. There are different ways to encode these addresses [4] and these worms can have different functionality. To illustrate the concept, in this section, we use all-destination encoding format with *broadcast/multicast* functionality. Later on, we explain about bit-string encoding. In the next section, we introduce alternative multidestination worm types like *gather* and *exchange*.

Once the worm is injected into the network, it is routed based on the address in the leading header flit corresponding to the first destination. As soon as the worm reaches the router of the first destination node, the flit containing this address is removed by the router. Now the worm is routed to the node whose address is contained in the next header flit (second destination in the ordered list). While the flits are being forwarded by the router of the first destination node to its adjacent router, they are also copied (consumed) flit-by-flit to the system buffer by the router-processor interface of the first destination. This process is carried out in each intermediate destination node of the ordered list. When the message reaches the last destination, it is not routed any further and gets completely consumed by the last destination node.

Fig. 1 illustrates an example of this mechanism with 3 neighboring destinations. A typical multidestination worm format with all destination encoding is shown in Fig. 1a. Figure 1b shows a snapshot of the system when flit A of the message has reached the router of node d3. It is assumed that the header flits have been appropriately consumed at the respective nodes prior to this step. Fig. 1c shows the modified system state when the worm moves ahead by one flit. Each flit is copied to the system buffer of the associated node and also propagated ahead to the next node. It is to be noted that such a mechanism is quite powerful and can deliver a message to multiple destinations much faster than using multiple unicast messages. This multidestination scheme is also quite general in the sense that unicast messages can always be implemented under this scheme as a subset operation with only one destination.

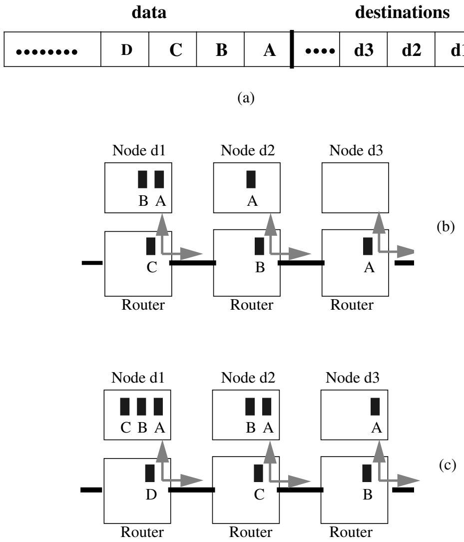

Figure 1: Multidestination wormhole mechanism with a multicast/broadcast worm: (a) a typical format of multidestination worm with all destination encoding, (b) snapshot of the system when flit A of the worm has reached the router of node d3 and (c) snapshot one clock cycle later.

2.2 Paths Conforming to Base Routing

Figure 2 shows examples of multidestination worms under *Base Routing Conformed Path* (BRCP) model, as proposed by us in [23]. For example, in an ecube system (assuming messages are routed first along row and then along column), a multidestination worm can cover a set of destinations in row/column/row-column order. It is to be noted that a set of destinations ordered in a column-row manner will be an *invalid* path under BRCP model for e-cube systems. Similarly, in a planar adaptive system, a multidestination worm can cover a set of destinations along any diagonal in addition to the flexibility supported by the ecube system. Such additional paths are shown as bold lines in the figure. If the underlying routing scheme supports west first turn model, it can provide further flexibility in covering a lot of destinations using a single worm. For this turn model example, a non-minimal west first routing scheme is assumed. If the base routing scheme supports non-minimal routing then the traversal of multidestination worms can also conform to non-minimal paths. Hence, this model is quite general and can be used by any routing scheme. A similar but restricted method of using only row-path and column-path in e-cube meshes has also been independently proposed by Boppana, Chalasani, and Raghavendra [3]. In this paper, we only use single-dimensional paths (row/column for 2D system and x/y/z for 3D system, etc.) without adaptivity to implement efficient global reduction operations.

2.3 Intrinsic Benefits

The significant benefit of the BRCP model comes from the fact that a message can pass through multiple destinations with the same overhead as that of sending it to a single destination, if the destinations can be grouped into a single worm under BRCP model. Even the simplest e-cube

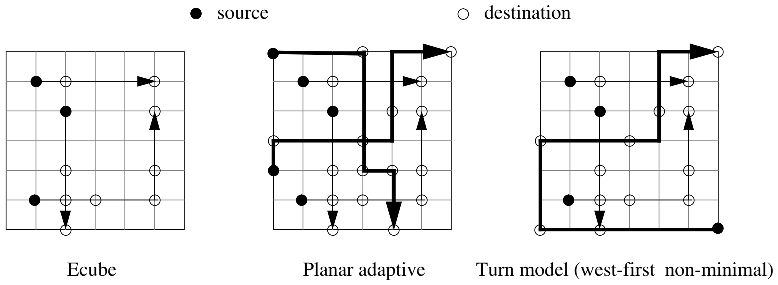

Figure 2: Examples of multidestination worms under BRCP model conforming to different base routing schemes in a 2-D mesh. In addition to ecube paths, added flexibility for new paths under planar and west first routing schemes are shown in bold lines.

routing scheme can take advantage of this model by grouping destinations on row, column, or row-column. As the adaptivity of the base routing scheme increases, more and more destinations can be covered by a single multidestination worm. In the next section, we use this fundamental property in introducing *gather* and *exchange* worms where information from multiple processors can be gathered or exchanged using a single communication start-up.

2.4 Bit-string Encoding of Multidestination Header

The example shown in Fig. 1 uses all-destination encoding. As the number of destination increases, such encoding makes the header quite long and increases the message size. Hence, it defeats the advantage of multidestination message passing. To alleviate this problem a *bit-string encoding* scheme has been proposed by Chiang and Ni [4]. For example, a message by P0 to cover processors P1, P2, and P5 in a 6-processor linear array system can have an address as 100110. As the message moves ahead, the appropriate bits are turned off by the intermediate routers. Finally, the message gets consumed by the router when the bit-string becomes all zero. This compact representation has significantly less overhead on message size. For a k-ary n-cube system with single-dimensional paths as discussed earlier, maximum k bits are sufficient to encode the destinations. The current generation systems support channel widths of 16-32 bits. Hence, maximum 1 or 2 flits are sufficient to encode destinations of a single dimensional path in a k-ary n-cube system with $k \leq 32$. Without loss of generality, in the following sections, we assume the multidestination header consisting of 1 flit only.

3 Reduction on Linear Array with Exchange Worms

In this section we introduce single-directional *gather* worm followed by bi-directional *exchange* worms. Next, we demonstrate the operational principle of exchange worms on a linear array to perform global reduction. The performance benefits of the proposed framework, its viability, and design considerations under the current technology are discussed.

3.1 Single-directional Gather Worm

The basic name indicates that this worm *gathers* information from multiple processors as it propagates. The *gather* operation can be any associative and commutative function (*sum, max, min,* or *user-defined*) [18]. Figure 3a illustrates a gather worm initiated by processor P5 with a bit-encoded address of 111111. A typical format of such a worm is shown in Fig. 3b. It consists of a message type (single destination vs. multiple destination), an *id*, a *function* field, a bit-string encoded multidestination address, and a packet of b bytes of data. We assume the required reduction operation to be element-wise operation on the data packet.

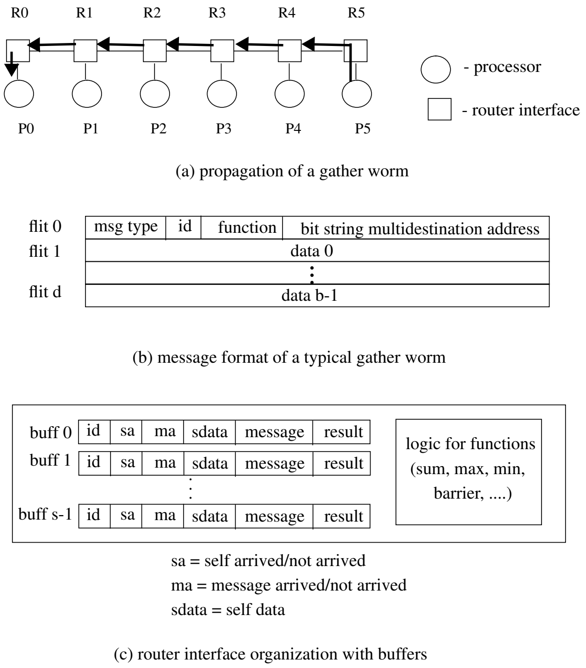

Figure 3: Implementing gather operation on a linear array with multidestination gather worms: (a) propagation of a gather worm, (b) message format of a gather worm, and (c) router interface organization.

Let us consider the movement of this worm assuming virtual cut-through wormhole-routing. Based on the multidestination address, the worm first gets routed to router R4 by R5. After reaching R4, this message is not consumed by processor P4 because the bit-string is still non-zero. It is stored into a buffer at the router interface R4. Similar to the concept of buffers/registers in a control network [6], each router interface consists of a set of buffers. Each buffer carries a few bits for *id*, a flag *sa* to indicate whether its associated processor has arrived at the gather point during its execution or not, a flag *ma* indicating whether the message for the corresponding id has arrived or not, a buffer *sdata* to hold the data supplied by its associated processor, a buffer *message* to hold the incoming message, and a buffer *result* to hold the result. A typical router interface organization with s such buffers is shown in Fig. 3c. These buffers can be accessed by the associated processor by memory-mapped I/O reference [14].

The worm, after arriving at router interface R4, waits for the flag *sa* to be 'on(1)'. If this flag is on, it indicates that the processor also has arrived at its *gather* execution point and has supplied data in the buffer *sdata*. Now the appropriate logic (indicated by the *function* field of the worm) gets activated and operates on *sdata* and *data* portion of the message to produce *result*. It may also happen that the processor has supplied its data to the router interface and made the flag *sa* 'on(1)'. However, the message has not arrived. In this case, the logic waits for the flag *ma* to be turned 'on(1)' after the arrival of the message at the interface.

Once the logic operation is over at R4, the worm is forwarded to the router R3 while replacing *data* of the message with the *result*. Like this, the worm moves ahead step by step while gathering results on its way. Finally, the message gets consumed by processor P0 and the gathered result is available with P0. The operation of a gather worm traversing k processors on a single-dimensional path can be expressed more formally as follows:

$$gather[0, k-1] = sdata_0 \odot sdata_1 \odot \cdots \odot sdata_{k-1}$$

where $sdata_i$ is the data item(s) associated with processor p_i, the operation $\odot$ specifies the required gather function and $gather[0, k-1]$ is the result gathered by the worm. While the gather operation is carried out, at each router interface, the *result* also gets computed and is available for the processor to use. Assume the gather worm is initiated by processor P0. Let the result available at processor P_i be $result_i$. Based on the operation of the gather worm, $result_i$ can be formally defined as follows:

$$result_i = sdata_0 \odot sdata_1 \odot \cdots \odot sdata_i$$

It is to be noted that this $result_i$ is nothing but prefix result of operation $\odot$ over data items *sdata* associated with processors.

3.2 Bidirectional Exchange Worm

It can be seen that after the gather operation, the intermediate results available at individual processors are prefix results and the final result is available with the last processor. If the reduction operation needs the result to be available only at an end processor then the one-step gather operation, as discussed above, is sufficient. However, it is not sufficient if the final result needs to be available at all processors. Of course, a multidestination *broadcast* worm as discussed in section 2.1 can be initiated by the end processor to distribute the final result to all processors on the array. However, this will take an additional communication step with its associated communication start-up and propagation time. In this section, we propose a new scheme where the second step can be avoided.

Instead of a single *gather* worm from one of the end processors, let us consider initiating a pair of *gather* worms from two end processors of the linear array. Both worms carry the same *id* and the same *function*. The bit-string addresses of the worms are appropriately reversed. Let us define such a pair of *gather* worms as *exchange* worms. Figure 4 shows a pair of positive and negative exchange worms for the linear array example being considered. The positive worm initiated by P0 carries a header 111110 and the negative worm by P5 carries 011111. Both worms follow the message format as shown in Fig. 3b. In order to support exchange worms, the router interface as shown in Fig. 3c needs minor modification. The single flag *ma* gets substituted by two flags (*pma* for positive worm and *nma* for negative worm). Similarly, the *message* buffer gets replaced by two buffers (*pmessage* and *nmessage*).

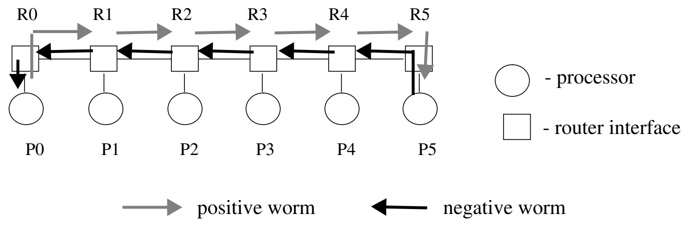

Figure 4: Implementing reduction operation on a linear array with a pair of positive and negative exchange worms.

At the beginning of a reduction operation, it is associated with a unique *id*. The end processors initiate the respective exchange worms. The intermediate processors make their respective *sdata* available at the router interface, make a copy of it as *result*, and turn the flag *sa* 'on(1)'. Since the movement of the worms can be asynchronous, the worms can arrive at a given router interface in any order. The logic at the router interface implements the following operations when the respective worms arrive. Since a common logic unit is involved the operations get sequentialized (the ordering does not matter) when both worms arrive simultaneously. The following operation is invoked when the positive exchange worm (*pexchange*) arrives at a router interface and the processor's data is available ($sa \wedge pma$ being true):

$$result_i = result_i \odot pexchange_{in}; \quad pexchange_{out} = sdata_i \odot pexchange_{in}$$

where $pexchange_{in}$ is the data coming into the router interface and $pexchange_{out}$ is the data going out of the interface. Similarly, when the negative exchange worm (*nexchange*) arrives and the processor's data is available ($sa \wedge nma$ being true), the following operation is carried out by the router interface:

$$result_i = result_i \odot nexchange_{in}; \quad nexchange_{out} = sdata_i \odot nexchange_{in}$$

where $nexchange_{in}$ is the data coming into the interface and $nexchange_{out}$ is the data going out of the interface. When the worms reach the respective end processors, they get consumed. When both worms have crossed a given router interface, the result is available to its associated processor for use. This leads to:

Lemma 1 *The operations described above for a pair of exchange worms implement global reduction on a linear array where the result is available to all processors.*

It can be seen that the above scheme just needs one communication step to perform complete (all processors participating) global reduction on a linear array. The number of communication steps is independent of the number of processors on a linear array. Using tree-based approach with unicast messages, the number of communication steps required are $2\lceil \log_2 k \rceil$, where k is the size of the linear array. This leads to:

Theorem 1 *Global reduction on a linear array of processors can be implemented in one communication step using a pair of multidestination exchange worms. With k processors on a linear array, the multidestination mechanism reduces the number of communication steps by a factor of* $2\lceil \log_2(d) \rceil$.

Besides the communication start-up time, for both unicast-based and exchange-worm based schemes, the overall reduction time depends on other parameters like node delay, path length, link propagation delay, and time to compute reduction operation at each router interface. In section 5, we take all these parameters into account and develop cost complexity models for both unicast-based and exchange worm-based schemes. Then we compare the models to derive performance of both schemes.

3.3 Design Considerations

The inherent benefits of the proposed scheme comes from the use of multidestination exchange worms. Such path-based implementations are prone to deadlock if careful consideration is not taken into account. In our earlier work [13], we had proposed *rendezvous* and *multirendezvous* primitives to achieve fast barrier synchronization for circuit-switched systems and wormhole-routed systems implementing Hamiltonian-path-based routing, respectively. The problem of deadlock for multiple concurrent barrier synchronization was pointed out in this work.

In this paper, we alleviate the deadlock problem by using virtual-cut through technique. The number of buffers available at each router interface determines the deadlock-freedom property. For complete reduction (all processors participating), two buffers are required at each router interface for the respective positive and negative exchange worms to arrive and reside. While implementing reduction with arbitrary number of processors, there can be maximum $(k/2)$ pairs of processor groups trying to perform different reduction operations concurrently (each processor can participate in maximum one operation at a given time). This leads to:

Lemma 2 *On a linear array with k processors, only 2 buffers are sufficient at each router interface to implement complete global reduction without deadlock. Arbitrary global reduction can be done without deadlock if k buffers are available.*

In the next section, we propose algorithms for k-ary n-cube systems using single-dimensional paths and the above result also holds true for these systems. The size of each buffer puts a restriction on the maximum data size on which the reduction operation can be carried out. In order to implement reduction on large size data, we present a pipelined algorithm in the next section. In section 5, we study the tradeoff of various buffer size on the overall cost of reduction operation. As we will see, the analysis indicates that as the communication start-up continue to decrease, 8-16 bytes buffers are sufficient to provide faster reduction operation using exchange worms. Even for large systems like 64x64 mesh, the above Lemma indicates that 32 buffers for each router are sufficient. The current generation system like IBM SP1/SP2 [25] already provides 1K bytes of buffer space at each router switch as central queue. Hence, our proposed exchange-worm based scheme is technically feasible in the current generation systems. With these design considerations, we move to next section proposing reduction algorithms for k-ary n-cube systems.

4 Reduction on k-ary n-cube

In this section, we formulate global reduction algorithms for k-ary n-cube systems based on the scheme developed for linear array in the last section. The algorithms are illustrated with respect to 2D mesh and then generalized for higher dimensional systems.

4.1 Two-dimensional System

Complete reduction can be achieved on a 2D mesh by considering it as a set of linear arrays. Based on the scheme proposed in Fig. 4, reduction can be achieved first along all rows in parallel. At the end of this step, each processor has the reduced result of its row processors. The second step involves similar exchange operation along the columns in parallel. During this step, each processor exchanges information along its column members and operates on the reduced result of the first step. By formulating the operations mathematically, as done in section 3.2, it can be easily verified that this two-step scheme implements global reduction over all processors with the final result being available to all processors. Figure 5 illustrates this two-step scheme.

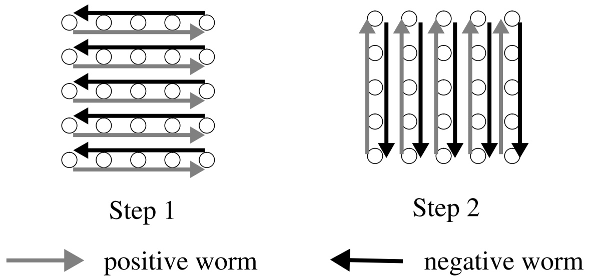

Figure 5: Complete global reduction on a 2D mesh using two step scheme. Both positive and negative exchange worms are used in each step.

The architectural supports, presented in the last section, assume fixed size buffers at the router interface. Let the maximum size of each buffer be b bytes. The above example illustrates that reduction operation on data size smaller than b bytes can be carried out in 2 steps on a 2D mesh. However, for larger data size, a *pipelined* version of the algorithm is needed. Assuming the message size is l bytes ($l > b$), it can be broken into $\lceil l/b \rceil$ packets. During every step along a dimension, the end processors, instead of initiating only one packet, initiate $\lceil l/b \rceil$ packets one after another. These packets move in a pipelined manner along a dimension. They are tagged with *sequence* numbers in addition to the *id* and the logic block at a router interface picks up packets with same sequence numbers to perform the reduction operation. Hence, for messages with large data size there will be $2\lceil l/b \rceil$ communication steps on a 2D mesh.

4.2 Higher Dimensional Systems

The above scheme can be easily extended to higher dimensional systems. For an n-dimensional system, each step (using a pair of exchange worms) can be repeated n times along different dimensions. For a 3D system, three steps will be needed to perform the operation along x, y, and z dimensions, respectively. It is to be noted that the ordering of these dimensions does not matter because the reduction operations being considered are associative and commutative. For k-ary n-cube system, the proposed scheme uses n communication steps. By using unicast-based scheme, the number of communication steps required in a k-ary n-cube system is $2\lceil \log_2 (k^n) \rceil$. This leads to:

Theorem 2 *Complete global reduction across k^n processors on a k-ary n-cube system can be implemented with n communication steps by using multidestination exchange worms for data size smaller than the buffer size available at a router interface. For larger data size, $n\lceil l/b \rceil$ communication steps are needed where l and b are data size and buffer size in bytes, respectively.*

As discussed in the introduction, in this paper, we emphasize only on complete reduction operation where all processors are participating. If arbitrary number of processors participate in the operation, then the exchange steps can not always be carried out. If some end processors do not participate then the structure of the problem (participating processors) becomes assymetric and it becomes difficult to carry out result from one phase to another phase. Such deficiency of path-based schemes has been analyzed in [21] while considering barrier synchronization on wormhole-routed systems using Hamiltonian path-based routing. Under these circumstances, one can use a two-phase method where the first phase involves single-directional gather worms to gather information along a dimension. After the final result is available at a processor, the second phase involves *multicast* operation using base-routing-conformed *broadcast/multicast* worms [23]. In [22] we have shown how to implement barrier synchronization for arbitrary set of processors using this two-phase scheme and hence, they are not repeated here.

5 Performance Analysis

In this section, first we develop cost complexity of both models for k-ary n-cube systems. Then we compare them for different system and technological parameters to see the effectiveness of the exchange worm-based scheme.

5.1 Cost Complexity

Let us consider the unicast-based scheme first. There are $2n \log_2 k$ communication steps. The upward-tree communication ($n \log_2 k$ steps) can be implemented dimension-wise. For example, on a 2D mesh, reduction operation can first be performed along x-dimension in $\log_2 k$ steps followed by along y-dimension. The steps along a dimension involve hops of distance $1, 2, 4, 8, \ldots, k/2$, respectively. During the upward tree communication, at the end of each level of communication, data reduction gets implemented on the entire message. However, during the broadcast phase, there is no reduction involved. This leads to the following total cost for implementing reduction on k-ary n-cube system using unicast-based scheme:

$$T_{uni} = n(2((\lceil log_2 k \rceil t_s) + (k-1+\lceil log_2 k \rceil)t_{node} + (k-1+(\lceil log_2 k \rceil \lceil l/w \rceil))t_p) + (\lceil log_2 k \rceil t_{compl})) \quad (1)$$

where t_s = communication start-up time, t_p = propagation time per link, t_{node} = node delay, t_{comp} = time to compute the reduction per element, l = message length in bytes, and w = channel width in bytes.

In exchange worm-based scheme, the buffer size at the router interface, as discussed in the last section, plays an important role in determining the overall cost. If data size is less than or equal to buffer size ($l \leq b$) then there is no pipelining and only one communication start-up is needed along each dimension. Hence, the overall cost of non-pipelined implementation of exchange worm-based scheme on k-ary n-cube system is:

$$T_{ex}^{np} = n(t_s + (kt_{node}) + ((\lceil l/w \rceil + 1)(k-1)t_p) + ((k-1)t_{comp}l)) \quad (2)$$

For larger data size, the pipelined algorithm as discussed in the last section is needed. Let the buffer size be b bytes. Under pipelining, the rate at which an exchange worm can move ahead along a dimension depends on the slowest of the three following factors: communication start-up time (t_s), time to transfer a buffer of data ($t_{node} + ((\lceil b/w \rceil) + 1)t_p$) through a hop, and time to perform reduction operation on a buffer of data (bt_{comp}). Let t_{max} denotes the maximum value of these three factors which reflects the rate at which the worms can move. The exchange worm-based scheme involves concurrent worms moving from two dimensions. For the pipelined version, it may happen that two opposite worms arrive at a router at the same time. This will happen if $\lceil l/b \rceil > k/2$. Under these circumstances, a packet may get delayed twice at a router interface for the reduction operation to be carried out. On the worst case scenario, a packet will encounter twice the overhead to perform reduction operation. Hence, for this situation we will define t_{max} to be $max(t_s, t_{node} + ((\lceil b/w \rceil) + 1)t_p, 2bt_{comp})$. This leads to the cost of pipelined algorithms as follows:

$$T_{ex}^{p} = n(\lceil l/b \rceil t_{max} + (kt_{node}) + ((\lceil b/w \rceil + 1)(k-1)t_p) + ((k-1)t_{comp}b)) \quad (3)$$

where t_{max} is appropriately modeled as discussed above.

Using the above models, we compared our scheme with the unicast-based scheme. The parameters were chosen to represent the current trend in technology. For overall evaluation, we assumed the following parameters: t_s (communication start-up time) as 0.5, 1.0 and 5.0 microseconds, t_p (link propagation time) as 5.0 nsec, t_{node} (router delay) as 20.0 nsec, t_{comp} (time per byte to compute a reduction operation) as 15.0 nsec. For scalability studies we used varying values for system and data size.

5.2 Overall Comparison for 2D Mesh

Figure 6 shows the comparison results for 16x16 and 32x32 mesh for two different communication start-up time: 5.0 microseconds and 1.0 microsecond. For the exchange worm-based scheme, we considered buffer sizes of 8, 16, and 32 bytes. A wide range of data sizes (1, 8, 16, 32, 64, 128, 256, and 512 bytes) were considered. It can be observed that with 5.0 microsecond start-up time, the exchange worm-based scheme performs superior for smaller data size ($< 128 bytes$). As the start-up time reduces to 1.0 microsecond, the scheme remains beneficial for even larger data sizes. For 1-flit data size (corresponding to barrier synchronization), the exchange worm-based scheme can implement a global barrier for 1024 processors on a 32x32 system in just 4.83 microsecond with a start-up time of 1.0 microsecond. For other system configurations and start-up time, a factor of 5-8 improvement was observed for exchange worm-based scheme over the unicast-based scheme to perform barrier synchronization.

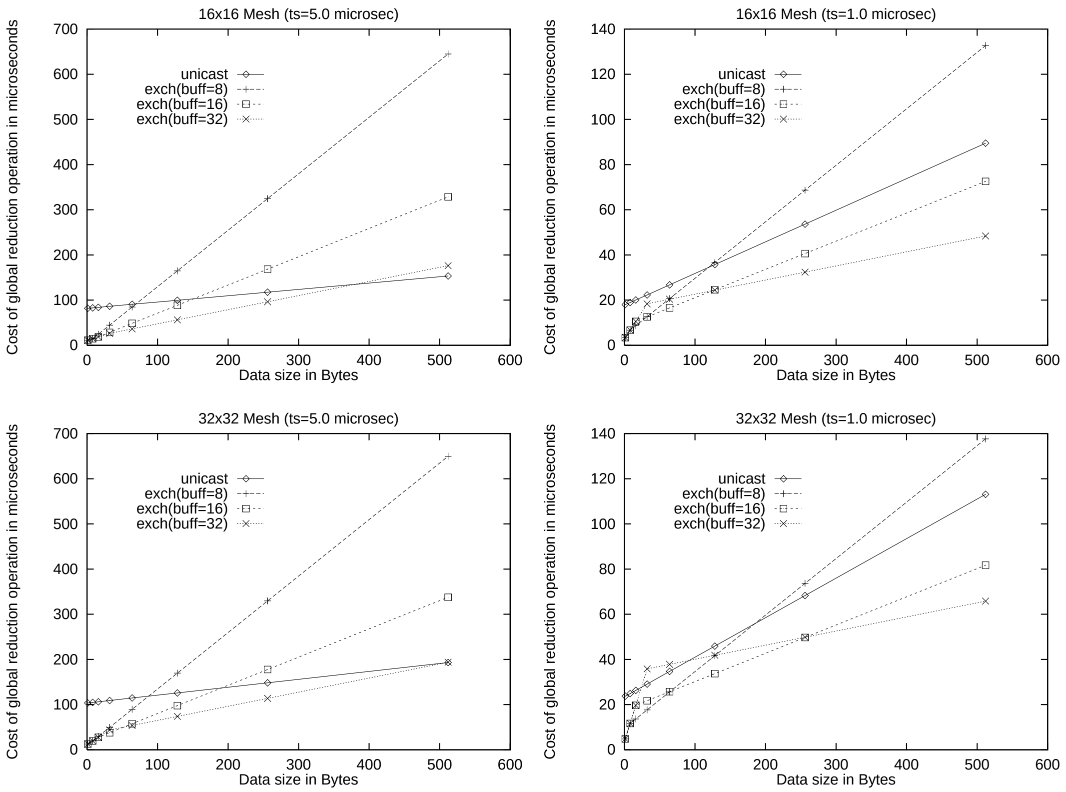

Figure 6: Comparison of unicast-based and exchange-worm based schemes to implement global reduction on 2D meshes with different data size (1-512 bytes), system size (16x16 and 32x32), communication start-up time (5.0 and 1.0 microseconds), and buffer size (8, 16, and 32 bytes).

The buffer size plays an important role in the overall cost of the reduction operation. With high communication start-up time, smaller buffer size introduces more number of start-ups and increases the cost. Hence, a larger buffer size is necessary for systems with high start-up time. However, when start-up time reduces, a configuration with larger buffer size makes the cost higher because the message propagation time starts dominating. Hence, it is suggested that smaller buffer size needs to be used for systems with low start-up time. Smaller buffer size leads to low cost implementation of the scheme and availability of more number of buffers at a router interface. As start-up time keeps on reducing for new generation systems, these results indicate that the proposed exchange worm-based scheme can be implemented with reduced cost.

5.3 Comparison for 3D Mesh and Effect of Communication Start-up

Figure 7 compares exchange worm-based scheme with unicast-based scheme for 10x10x10 system with various start-up time. It can be observed that as the start-up time reduces, the exchange worm-based scheme demonstrates consistent superiority for all data sizes. With a start-up time of 0.5 microsecond, it just takes 2.78 microseconds to barrier synchronize 1K processors using exchange worm-based scheme compared to 14.13 microseconds by using unicast-based scheme. For start-up time with 1.0 and 5.0 microseconds, the factors of improvement are 6.1 and 7.5, respectively. With 0.5 microsecond start-up time, a global reduction operation on 512 bytes of data can be implemented in 55.0 microseconds using exchange worms compared to 121.35 microseconds needed for unicast-based scheme. As the start-up time reduces, buffer size of 16 bytes seems to be optimal for all data sizes.

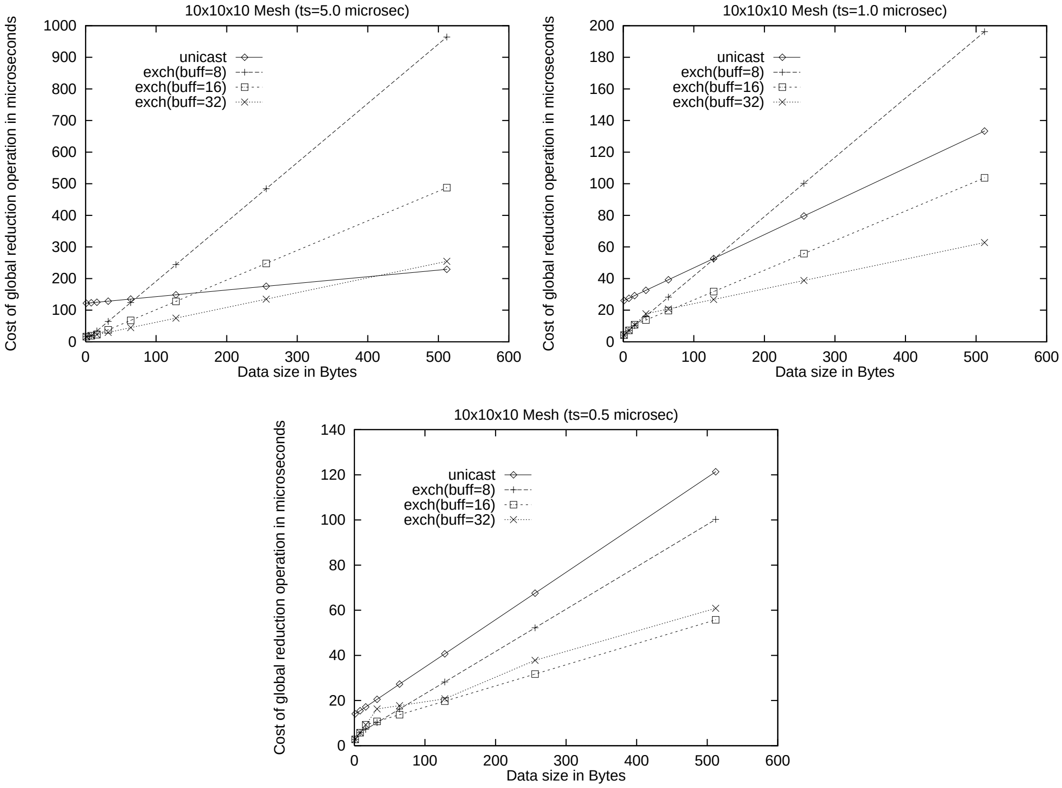

Figure 7: Comparison of unicast-based and exchange-worm based schemes to implement global reduction on 10x10x10 mesh with different data size (1-512 bytes), communication start-up time (5.0, 1.0, and 0.5 microseconds), and buffer size (8, 16, and 32 bytes).

5.4 Scalability with System Size

To study how scalable the exchange worm-based scheme is with respect to unicast-based scheme, we studied the effect of reduction operation for 32 and 256 bytes of data size on various system configurations (4x4, 8x8, 16x16, and 32x32). Figure 8 shows the comparison. It can be observed that with appropriate buffer size, as system size increases, the exchange worm-based scheme continues to perform reduction operation with less cost. For smaller data size (32 bytes), the improvement varies from 1.7-2.1 and for large data size (256 bytes) it varies between 1.9-5.0. Such performance improvement demonstrates the scalability with respect to increase in system and data size.

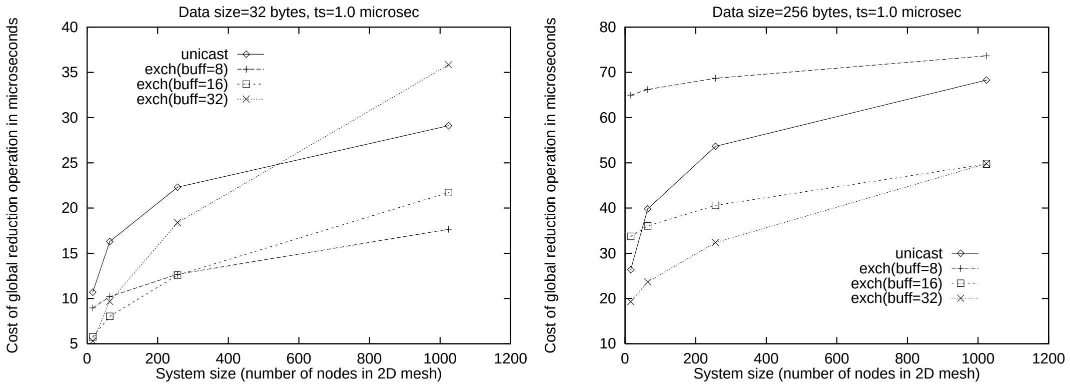

Figure 8: Scalability of exchange-worm based scheme over unicast-based scheme with increase in system size in 2D meshes: (a) data size=32 bytes and (b) data size=256 bytes.

6 Conclusions and Future Research

In this paper we have presented a new approach to implement fast and scalable global reduction (including barrier synchronization) in k-ary n-cube wormhole systems using multidestination exchange worms. The necessary architectural supports and design modifications to the router interface to implement the operations are presented. Algorithms are developed and evaluated for reduction operations on k-ary n-cube systems for various data sizes. It is shown that only $n\lceil l/b \rceil$ communication steps are needed with exchange worms to implement reduction on l bytes of data with a buffer size of b bytes at a router interface. Compared to the unicast-based scheme, the proposed framework demonstrates an asymptotic reduction of $2\log_2 k$ communication steps for smaller data size.

Analytical results indicate that the proposed scheme is far superior compared to to the unicast-based scheme for a wide variety of system topology, system size, data size, and communication start-up time. With a communication start-up time of 1.0 microsecond, the scheme has potential to barrier synchronize 1K processors, organized as a 32x32 mesh, in just 4.83 microseconds compared to 23.75 microseconds required by the unicast-based scheme. It is shown that the performance of the scheme is sensitive to the buffer size at a router interface. As the communication start-up time continues to decrease in current and future generation systems, it is shown that a few buffers of 8-16 bytes each are sufficient to take advantage of the scheme. Scalability studies with respect to system size and data size indicate that the new approach always demonstrates superiority over the unicast-based scheme and hence, makes the framework suitable for current and future generation wormhole systems.

In this paper, we have studied the usage of multidestination mechanism for global reduction operation. We are extending our work to other collective communication patterns like scatter, complete-exchange, and parallel-prefix. As the system size grows the proposed scheme encounters more delay due to increased path length. We are working on alternative schemes to reduce such impact of path length. Since barrier synchronization is a basic mechanism for implementing distributed shared memory on direct networks, we are extending this framework to see how scalable distributed-shared-memory systems can be designed on wormhole direct networks with low synchronization overheads.

References

[1] M. Barnett, S. Gupta, D. G. Payne, L. Shuler, R. van de Geijn, and J. Watts. Interprocessor Collective Communication Library (Intercom). In *Scalable High Performance Computing Conference*, pages 357–364, 1994.

[2] M. Barnett, R. Littlefield, D. G. Payne, and R. Van de Geijn. Global Combine on Mesh Architectures with Wormhole Routing. In *Proceedings of the International Parallel Processing Symposium*, pages 156–162, 1993.

[3] R. V. Boppana, S. Chalasani, and C. S. Raghavendra. On Multicast Wormhole Routing in Multicomputer Networks. In *Symposium on Parallel and Distributed Processing*, 1994. to appear.

[4] C.-M. Chiang and L. M. Ni. Multi-Address Encoding for Multicast. In *Proceedings of the Parallel Computer Routing and Communication Workshop*, 1994. to appear.

[5] A. A. Chien and J. H. Kim. Planar-Adaptive Routing: Low-Cost Adaptive Networks for Multiprocessors. In *Proceedings of the International Symposium on Computer Architecture*, pages 268–277, 1992.

[6] Cray Research, Inc. *Cray T3D System Architecture Overview*, 1993.

[7] W. J. Dally. Virtual Channel Flow Control. *IEEE Transactions on Parallel and Distributed Systems*, pages 194–205, Mar 1992.

[8] W. J. Dally and C. L. Seitz. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Transactions on Computers*, pages 547–553, May 1987.

[9] R. A. Van de Geijn. On Global Combine Operations. *Journal of Parallel and Distributed Computing*, 22:324–328, 1994.

[10] J. Duato. A New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks. *IEEE Transactions on Parallel and Distributed Systems*, 4(12):1320–1331, 1993.

[11] A. Feldmann, T. Gross, D. O'Hallaron, and T. Stricker. Subset Barrier Synchronization on a Private Memory Parallel System. In *Proceedings of the Symposium on Parallel Algorithms and Architectures*, pages 209–218, 1992.

[12] K. Ghose and D.-C. Cheng. A High Performance Barrier Synchronizer and Its Novel Applications in Highly Parallel Systems. In *Symposium on Parallel and Distributed Processing*, pages 616–619, 1991.

[13] S. K. S. Gupta and D. K. Panda. Barrier Synchronization in Distributed-Memory Multiprocessors using Rendezvous Primitives. In *Proceedings of the International Parallel Processing Symposium*, pages 501–506, April 1993.

[14] J. L. Hennessy and D. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 1990.

[15] Intel Corporation. *Paragon XP/S Product Overview*, 1991.

[16] S. Lennart Johnsson. Issues in High Performance Computer Networks. *Special Issue of IEEE TCCA Newsletter on Interconnection Networks for High Performance Computing Systems*, Fall 1994.

[17] P. K. Mckinley, Y.-J. Tsai, and D. F. Robinson. A Survey of Collective Communication in Wormhole-Routed Massively Parallel Computers. Technical Report MSU-CPS-94-35, Dept. of Computer Science, Michigan State University, 1994.

[18] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*, Mar 1994.

[19] L. Ni and P. K. McKinley. A Survey of Wormhole Routing Techniques in Direct Networks. *IEEE Computer*, pages 62–76, Feb. 1993.

[20] M. T. O'Keefe and H. G. Dietz. Hardware Barrier Synchronization: Static Barrier MIMD (SBM). In *Proceedings of the International Conference on Parallel Processing*, pages I: 35–42, Aug 1990.

[21] D. K. Panda. Optimal Phase Barrier Synchronization in k-ary n-cube Wormhole-routed Systems using Multirendezvous Primitives. In *Workshop on Fine-Grain Massively Parallel Coordination*, pages 24–26, May 1993.

[22] D. K. Panda. Fast Barrier Synchronization in Wormhole k-ary n-cube Networks with Multidestination Worms. In *International Symposium on High Performance Comuter Architecture*, 1994. submitted.

[23] D. K. Panda, S. Singal, and P. Prabhakaran. Multidestination Message Passing Mechanism Conforming to Base Wormhole Routing Scheme. In *Proceedings of the Parallel Computer Routing and Communication Workshop,*, 1994.

[24] J. A. Solworth and J. Stamatopoulos. Integrated Network Barriers for D-dimensional Meshes. In *Proceedings of the Working Conference on Architectures and Compilation Techniques for Fine and Medium Grain Parallelism*, 1992.

[25] Craig Stunkel, D. Shea, D. G. Grice, P. H. Hochschild, and M. Tsao. The SP1 High Performance Switch. In *Scalable High Performance Computing Conference*, pages 150–157, 1994.

[26] H. Xu, P. K. McKinley, and L. Ni. Efficient Implementation of Barrier Synchronization in Wormhole-routed Hypercube Multicomputers. *Journal of Parallel and Distributed Computing*, 16:172–184, 1992.